Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-milestone-2-trivia-2024-m24/grade/vvh

Course: IT114-003-F2024
Assigment: [IT114] Milestone 2 Trivia 2024 (M24)
Student: Valeria C. (vvh)

Submissions:

Submission Selection

1 Submission [submitted] 11/11/2024 10:03:07 PM

Instructions

COLLAPSE

1. Implement the Milestone 2 features from the project's proposal document: https://docs.google.com/document/d/1h2aEWUoZ-etpz1CRI-StaWbZTjkd9BDMq0b6TXK4utI/view
2. Make sure you add your ucid/date as code comments where code changes are done
3. All code changes should reach the Milestone2 branch
4. Create a pull request from Milestone2 to main and keep it open until you get the output PDF from this assignment.
5. Gather the evidence of feature completion based on the below tasks.
6. Once finished, get the output PDF and copy/move it to your repository folder on your local machine.
7. Run the necessary git add, commit, and push steps to move it to GitHub
8. Complete the pull request that was opened earlier
9. Upload the same output PDF to Canvas

Branch name: Milestone2

Group

Group: Payloads
Tasks: 3
Points: 1

COLLAPSE

Task

100%

Group: Payloads
Task #1: Base Payload Class
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:
All code screenshots must have ucid/date visible.

Columns: 2

Sub-Task

100%

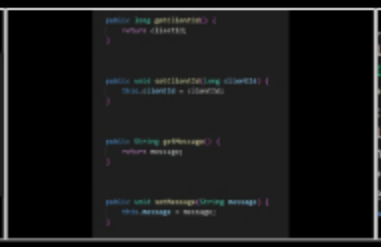
Group: Payloads
Task #1: Base Payload Class
Sub Task #1: Show screenshot of the code for this file

Task Screenshots

Gallery Style: 2 Columns

4 2 1

screenshot of the payloads.java code 1st part



screenshot of the payloads.java code 2nd part

screenshot of the payloads.java code 3rd part

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Sub-Task

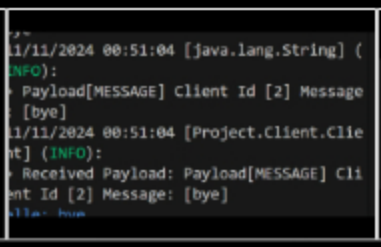
100%

Group: Payloads
Task #1: Base Payload Class
Sub Task #2: Show screenshot examples of the terminal output for this object

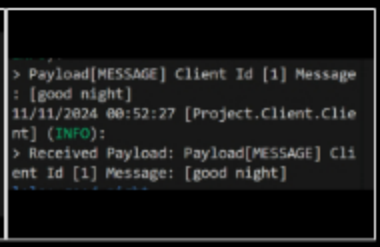
Task Screenshots

Gallery Style: 2 Columns

4 2 1



example of the terminal output when clients send a message



another example demonstrating payloads in action

screenshot between clients

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the purpose of each property and serialization

Response:

The Base Payload Class is designed to facilitate data

transfer between the client and server. The clientId property identifies each client, ensuring data is directed to the correct recipient, while the payloadType property specifies the kind of data being transmitted. the message property holds the main content of each payload, such as text or commands. serialization is applied to this class to convert it into a format suitable for network transmission or file storage allowing the class's complex structure to be converted into a data stream that can be sent over a network or saved, making it easy to reconstruct the original object on the other side. serialization is useful here for transmitting data between clients and servers and storing data for saving logs, session information, or game states.

End of Task 1

Task

100%

Group: Payloads
Task #2: QAPayload Class
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:
All code screenshots must have ucid/date visible.

Columns: 2

Sub-Task

100%

Group: Payloads
Task #2: QAPayload Class
Sub Task #1: Show screenshot of the code for this file

Sub-Task

100%

Group: Payloads
Task #2: QAPayload Class
Sub Task #2: Show screenshot examples of the terminal output for this object

Task Screenshots

Gallery Style: 2 Columns

4 2 1

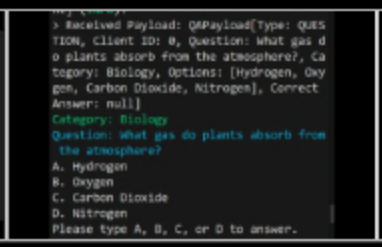
code image for QAPayload class part1

code image for QAPayload class part2

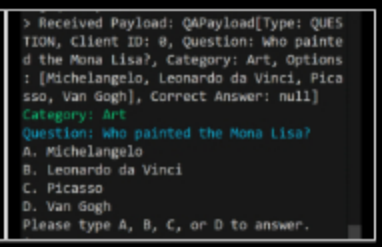
Task Screenshots

Gallery Style: 2 Columns

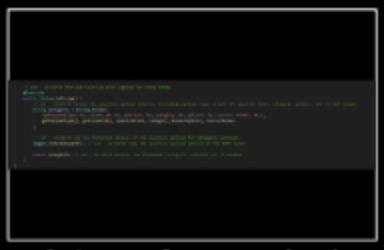
4 2 1



example displaying the terminal output of QAPayload



another example of QAPayload in action



code image for QAPayload class part3

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the purpose of each property

Response:

The QAPayload class represents a type of Payload for handling trivia questions and options. The questionText property holds the text of the trivia question, providing the main content of each question prompt. The category property specifies the category of the question allowing questions to be filtered by topic. answerOptions is a list of possible answers, presented as options to the user, while correctAnswer stores the correct option indicator such as "A", "B"... clientId inherited from Payload identifies the client associated with this question, helping to track responses. Serialization allows instances of QAPayload to be converted into a format suitable for network transmission. The toString() method provides a structured view of all properties.

End of Task 2

Task

100%

Group: Payloads
Task #3: PointsPayload Class
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:
All code screenshots must have ucid/date visible.

Columns: 2

Sub-Task Group: Payloads

Sub-Task Group: Payloads

100%

Task #3: PointsPayload Class
Sub Task #1: Show screenshot of the code for this file

100%

Task #3: PointsPayload Class
Sub Task #2: Show screenshot examples of the terminal output for this object

Task Screenshots

Gallery Style: 2 Columns

4 2 1

PointsPayload.java code part1

PointsPayload.java code part2

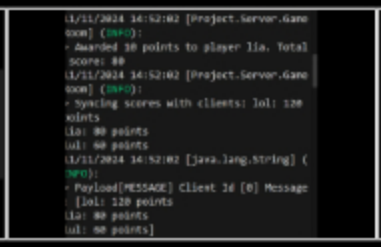
Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

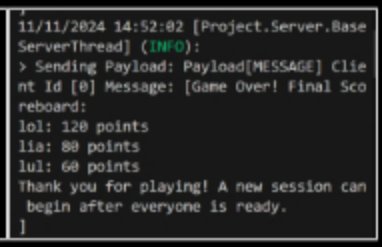
Task Screenshots

Gallery Style: 2 Columns

4 2 1



record of points based on answers from clients during the trivia game



"scoreboard" with client name and points gained at the end of the game

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the purpose of each property

Response:

The PointsPayload class is designed to manage and transmit player scores in a game. playerId or clientId identifies each player uniquely, ensuring scores are correctly assigned to the right individual. the score property holds the current score of a player, which may be updated as the game progresses. an optional leaderboard or playerScores property could store a collection or map of all players' scores, allowing the system to track and display the standings in real-time. these properties together enable PointsPayload to convey up-to-date score data across the server and clients, providing a clear view of each player's progress.

End of Task 3

End of Group: Payloads
Task Status: 3/3

Group

100%

Group: Questions
Tasks: 1
Points: 1.25

COLLAPSE

Task

100%

Group: Questions
Task #1: Data Structure
Weight: ~100%
Points: ~1.25

COLLAPSE

Details:
All code screenshots must have ucid/date visible.

Columns: 2

Sub-Task

100%

Group: Questions
Task #1: Data Structure
Sub Task #1: Show the code related to Questions/Answers (text, category, answers, correct)

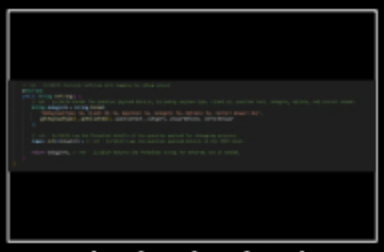
Task Screenshots

Gallery Style: 2 Columns

4 2 1

QAPayload code related to the questions/answers part 1

QAPayload code related to the questions/answers part 2



QAPayload code related to the questions/answers part 3

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain the class and each property

Response:

The QAPayload class represents a Payload type for managing trivia questions and answers. This class is

Sub-Task

100%

Group: Questions
Task #1: Data Structure
Sub Task #2: Show the text file with your sample questions and answers

Task Screenshots

Gallery Style: 2 Columns

4 2 1

this is my text file containing the questions organized by text, category, answers, and the correct one

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

designed to transmit question-related data from the server to clients and evaluate responses. It includes four main properties specific to quiz questions questionText which stores the actual trivia question text that will be shown to users. category which represents the category of the trivia question, which can help users understand the context (e.g., "Science," "History"). answerOptions which holds a list of possible answers, allowing clients to display multiple choices for each question. correctAnswer which indicates the correct answer among the options, aiding in response validation and scoring. the class also includes a LoggerUtil instance for structured logging, enabling clear debug output when handling question data. the constructor initializes these fields and sets the payload type to QUESTION. This class implements toString() to format the payload details in a readable way and logs them using LoggerUtil for easier debugging.

End of Task 1

End of Group: Questions
Task Status: 1/1

Group

100%

Group: Session Start
Tasks: 2
Points: 1.25

COLLAPSE

Task

100%

Group: Session Start
Task #1: Question List
Weight: ~50%
Points: ~0.63

COLLAPSE

Details:
All code screenshots must have ucid/date visible.

Sub-Task

100%

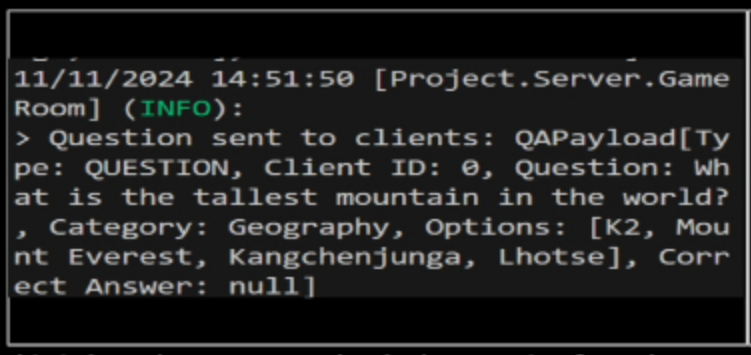
Group: Session Start
Task #1: Question List

Sub Task #1: Show the GameRoom loading the question list from a file

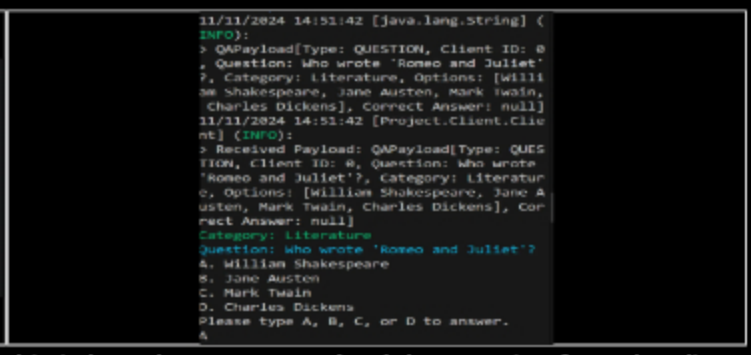
Task Screenshots

Gallery Style: 2 Columns

4 2 1



this is how the gameroom loads the question from the server side



this is how the gameroom load the question from the client side

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1. File loading (data input): The loadQuestionsFromFile method reads questions from a file, where each line represents a question and its attributes (question text, category, answer options, and correct answer). each line is split into parts, which are used to create a QAPayload object (representing a trivia question). this QAPayload object is stored in a list (questions) for later access during gameplay.
2. session start and question display: when the session starts (onSessionStart), the program first calls loadQuestionsFromFile to populate the list of questions from the file. It then optionally calls displayLoadedQuestions to log each loaded question to the console, providing a way to verify that questions are loaded correctly. 3.Round Management:the startRound method is triggered after loading the questions, beginning a new round by randomly selecting a question from the list. This selected question is removed from the list to ensure it doesn't repeat in future rounds.the chosen question is then sent to all connected clients via the sendQuestionToClients method. This method sends the QAPayload (without revealing the correct answer) to each client.
3. Answer Processing:as clients respond, their answers are sent back to the server, where they are handled by processAnswer. this method verifies whether the player's answer matches the correctAnswer of the QAPayload. If correct, points are awarded to the player's score. if all players answer, the round ends early, triggering onRoundEnd.
4. End of Round: onRoundEnd synchronizes the leaderboard across clients, and checks if more rounds are needed. If the maximum rounds are completed or questions are exhausted, endSession is called to display the final scoreboard and reset the game.
5. Session Reset: the endSession method displays the final scores to each client and resets all players' scores and statuses, preparing the game for a new session.

End of Task 1

Task

100%

Group: Session Start
Task #2: First round
Weight: ~50%
Points: ~0.63

COLLAPSE

Details:
All code screenshots must have ucid/date visible.

Sub-Task

100%

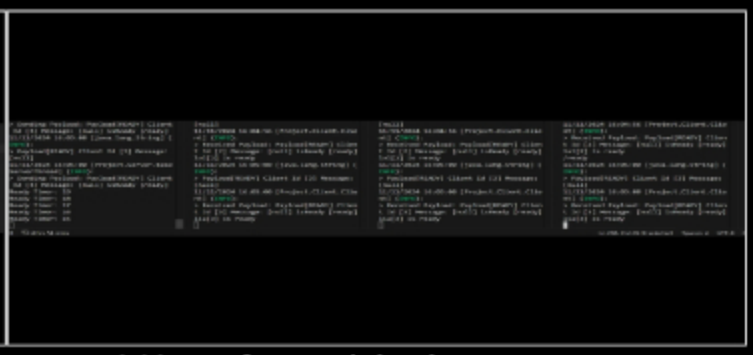
Group: Session Start
Task #2: First round
Sub Task #1: Show the code that triggers the first round

Task Screenshots

Gallery Style: 2 Columns

4 2 1

this is how it looks in the gameroom.java code



program initiates after /ready has been sent

This is how first questions displays randomly to the client

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

End of Task 2

End of Group: Session Start
Task Status: 2/2

Group

100%

Group: Round Start
Tasks: 3
Points: 1.25

COLLAPSE

Task

100%

Group: Round Start
Task #1: Picking a question
Weight: ~33%
Points: ~0.42

COLLAPSE

Sub-Task

100%

Group: Round Start
Task #1: Picking a question
Sub Task #1: Show the code related to getting a random question from the list (it should get removed so it's not used again)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

this is how startRound() look in gameroom which displays questions to the clients ensuring that they dont repeat between rounds

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1.check for questions:before starting a round, check if there are any questions left in the questions list. if no questions are available, end the session (e.g., by calling onSessionEnd()). 2.select random question:use a random index to select a question from the list. This ensures that each round has a different question. 3.remove the selected question:immediately remove the selected question from the questions list. This prevents it from being used again in future rounds. 4.prepare for new round:clear the previous round's answers stored in playerAnswers to avoid mixing answers between rounds. 5.send question to clients:send the selected question to all connected clients, allowing

them to view and respond to it. 6.log the selection:log details of the selected question for debugging and verification purposes. 7.manage timing:reset and start a timer for the round. this timer will automatically end the round when time runs out or when all players have answered

End of Task 1

Task

100%

Group: Round Start
Task #2: Syncing Question
Weight: ~33%
Points: ~0.42

COLLAPSE

Details:
All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.

Columns: 2

Sub-Task

100%

Group: Round Start
Task #2: Syncing Question
Sub Task #1: Show the code the syncs the question and answer choices to all clients (don't send the correct answer)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

sendQuestionToClients which shows the synsc the question and answer choices to all clients in gameroom.java

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Sub-Task

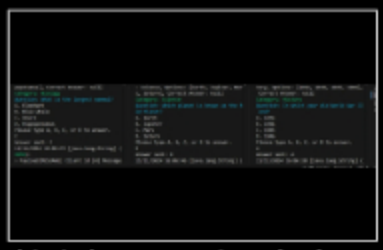
100%

Group: Round Start
Task #2: Syncing Question
Sub Task #2: Show the terminal output of all clients receiving it

Task Screenshots

Gallery Style: 2 Columns

4 2 1



this is how questions look between clients

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1.server prepares questions:the GameRoom class loads

questions from a file into a list (questions), each represented by a QAPayload object containing the question text, category, options, and correct answer. 2.server selects a random question:at the start of each round, the server randomly selects a question from this list using startRound.the selected question is removed from the list to prevent repetition. 3.server syncs question to clients:the sendQuestionToClients method in GameRoom is called to send the selected question to all connected clients.this method creates a new QAPayload object without the correct answer (to avoid revealing it).the server sends this payload to each client using sendQuestion. 4.client receives and displays question:on each client, the displayQuestion method in the Client class receives the question payload.this method outputs the question text, category, and answer options (A, B, C, D) in a user-friendly format, excluding the correct answer. 5.client responds:the client is prompted to type an answer (A, B, C, or D).the response is sent back to the server to be processed. 6.server processes answers:the server collects each client's response, checks correctness, updates scores, and tracks if all clients have answered. 7.once all clients answer, or time expires, the round ends, and the next round or end of the game triggers.

End of Task 2

Task

100%

Group: Round Start
Task #3: Round Timer
Weight: ~33%
Points: ~0.42

COLLAPSE

Details:
All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.

Columns: 3

Sub-Task

100%

Group: Round Start
Task #3: Round Timer
Sub Task #1: Show the code that triggers the round

Sub-Task

100%

Group: Round Start
Task #3: Round Timer
Sub Task #2: Show how the clients are informed of the

Sub-Task

100%

Group: Round Start
Task #3: Round Timer
Sub Task #3: Show an example from the terminal

timer (including the

time (i.e., synced or

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code that start the round time

code to reset the round timer

code to trigger the round timer to start of each round

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1. resetRoundTimer:roundTimer is set to a new TimedEvent with a duration of 30 seconds. this::onRoundEnd is passed as a callback, so when 30 seconds elapse, onRoundEnd is automatically triggered to end the round.
2. startRoundTimer: ensure only one timer is active at a time, the resetRoundTimer method is used to cancel any existing timer before starting a new one.
3. onRoundStart: started at the beginning of each round within the onRoundStart method, which calls both resetRoundTimer and

Task Screenshots

Gallery Style: 2 Columns

4 2 1

here is the code that should show clients the time when there are 5 second left

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1. initialize timer on server: a countdown timer (TimedEvent) starts on the server with a set duration (e.g., 30 seconds).
2. server logs countdown: every second, the timer's tick callback updates and logs the remaining time on the server.
3. notify clients at key intervals:the server only sends a message to clients when time reaches a critical threshold (e.g., the last 5 seconds). 4.timer expires: when time reaches 0, the expireCallback triggers onRoundEnd(), which ends the round and stops the timer. 5.client receives notifications: bclients see periodic updates only on critical moments

Task Screenshots

Gallery Style: 2 Columns

4 2 1

example from the terminal showing time countdown on the server side

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

startRoundTimer to initialize the timer.

End of Task 3

End of Group: Round Start
Task Status: 3/3

Group

100%

Group: During Round
Tasks: 2
Points: 1.5

COLLAPSE

Task

100%

Group: During Round
Task #1: Answer Command
Weight: ~50%
Points: ~0.75

COLLAPSE

Details:
All code screenshots must have ucid/date visible.

Columns: 4

Sub-Task 100% Group: During Round Task #1: Answer Command Sub Task

Sub-Task 100% Group: During Round Task #1: Answer Command Sub Task

Sub-Task 100% Group: During Round Task #1: Answer Command Sub Task

Sub-Task 100% Group: During Round Task #1: Answer Command Sub Task

Task Screenshots

Gallery Style: 2 Columns

4 2 1

client.java

Task Screenshots

Gallery Style: 2 Columns

4 2 1

gameroom.java

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code part

Task Screenshots

Gallery Style: 2 Columns

4 2 1

client/server

shows client

showing the code that processes the answer text

Caption(s) (required) ✓
Caption Hint:
Describe/highlight what's being shown

Task Response Prompt

Explain in concise steps how this logically works
Response:

1. user input:the client waits for the user to type an input in the console (e.g., answer A or simply A).
2. command recognition:the client reads the input and checks if it starts with answer.if it does, it extracts the answer choice (e.g., A, B, C, or D) from the command.
3. payload creation:the client packages this answer choice in an AnswerPayload object, which includes the client ID and selected answer.
4. sending payload to server:the client sends the AnswerPayload to the server through

show that the answer is blocked from changing

Caption(s) (required) ✓
Caption Hint:
Describe/highlight what's being shown

Task Response Prompt

Explain in concise steps how this logically works
Response:

1. fetching the player:retrieves the ServerPlayer instance from playersInRoom using clientId.
2. blocking multiple submissions:if the player does not exist (player == null) or if playerAnswers already contains an entry for this clientId, the answer submission is blocked with a warning log, and the method returns early. this prevents the player from submitting an answer more than once.
3. correctness check:the answer is compared to currentQuestion.getCorrectAnswer() If the answer is correct, incrementScore() adds 10 points to

where it tells all players a player locked in an answer

Caption(s) (required) ✓
Caption Hint:
Describe/highlight what's being shown

Task Response Prompt

Explain in concise steps how this logically works
Response:

1.retrieve and validate player:look up the player by clientId.If the player is null or has already submitted an answer (exists in playerAnswers), log a warning and exit (this prevents answer changes). 2.check correctness:compare the submitted answer with the correct answer stored in currentQuestion.store the result (true for correct, false for incorrect) in playerAnswers, marking this player's answer as locked. 3.broadcast answer lock-in:construct a message (lockInMessage) indicating the player has locked in their answer.send lockInMessage to all players, informing them that this player has submitted their answer. 4.update score (if correct):if the answer is correct, award points (e.g., player.incrementScore(10)) and log the update. 5.end round early if all players

output code

score between clients

Caption(s) (required) ✓
Caption Hint:
Describe/highlight what's being shown

an established connection.

5. server processing:the server receives the payload, identifies it as an answer, and processes it. It checks the answer's correctness, logs it, and updates the player's score accordingly.
6. client feedback:after processing, the server might send feedback (e.g., "Correct!" or "Incorrect!") or a scoreboard update to all clients.

the player's score.
4.early round end:if all players have submitted their answers (playerAnswers.size() == playersInRoom.size()), the round ends immediately using onRoundEnd().

have answered:if all players have submitted answers (i.e., playerAnswers.size() matches the total players), call onRoundEnd() to proceed to the next phase.

End of Task 1

Task

100%

Group: During Round
Task #2: Join in progress
Weight: ~50%
Points: ~0.75

COLLAPSE

Details:
All code screenshots must have ucid/date visible.

Sub-Task

100%

Group: During Round
Task #2: Join in progress
Sub Task #1: Show the code related to syncing game state to a joining client if the game is in progress

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code related to syncing game state to a joining client if the game is in progress

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1.client joins room: when a new client joins the GameRoom, the onClientAdded() method is triggered. 2.sync game phase: the syncCurrentPhase() method sends the current phase of the game (e.g., READY, IN_PROGRESS, or FINISHED) to the new client, so they know the game status. 3.sync player status: the syncReadyStatus() method sends the "ready" status of all players to the new client, informing them of who is ready to play. 4.check game progress: if the game is in progress (Phase.IN_PROGRESS), the server proceeds to send additional game data to the joining client. 5.send current question: the syncCurrentQuestion() method sends the current question to the new client. This includes the question text, category, and answer options, but excludes the correct answer to avoid giving away information. 6.send current scores: the syncPlayerScores() method sends the scoreboard to the new client, showing them the points each player has accumulated so far.

End of Task 2

End of Group: During Round
Task Status: 2/2

Group

100%

Group: Round End
Tasks: 3
Points: 1.5

COLLAPSE

Task

100%

Group: Round End
Task #1: End Round Conditions
Weight: ~33%
Points: ~0.50

COLLAPSE

Details:
All code screenshots must have ucid/date visible.

Columns: 2

Sub-Task
100%
Group: Round End
Task #1: End Round Conditions
Sub Task #1: Show the code related to ending a round when all Players answer correctly

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code related to ending a round when all players answer correctly

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works
Response:

1.player submits answer:when a player submits an answer, the processAnswer() method is called. 2.answer validation:the method checks if the player is valid (exists in the game) and hasn't already submitted an answer. If the player has already answered, the method exits, preventing duplicate answers. 3.check answer correctness:the submitted answer is compared to the correct answer for the current question.If correct, the player's score is increased by 10 points. 4.record answer:the player's answer (correct or incorrect) is recorded in playerAnswers, marking them as having answered. 5.check all players' responses:the method checks if all players have answered by comparing playerAnswers.size() to playersInRoom.size().if all players have answered, onRoundEnd() is called to immediately end the round. 6.end of round:onRoundEnd() performs tasks such as awarding points, updating clients with scores, and preparing for

Sub-Task
100%
Group: Round End
Task #1: End Round Conditions
Sub Task #2: Show the code related to ending a round when the round timer expires

Task Screenshots

Gallery Style: 2 Columns

4 2 1

showing the code related to ending a round when the round timer expires part 1

showing the code related to ending a round when the round timer expires part 2

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works
Response:

1.round timer initialization:startRoundTimer() sets up a TimedEvent countdown (e.g., 30 seconds).this timer is configured to call onRoundEnd() automatically when it expires. 2.timer expiration:when the timer reaches zero, it triggers onRoundEnd() through its expireCallback. 3.ending the round:onRoundEnd() does the following:awards points to players based on their answers.Updates all players with the latest scores. 4.clears player answers to prepare for the next round.Increments the round count. 5.check for game end:if the maximum number of rounds has been reached or there are no more questions, endSession() is called to end the game.Otherwise, it starts the next round by calling onRoundStart()

the next round or ending the session if all rounds are complete.

End of Task 1

Task

100%

Group: Round End
Task #2: Points
Weight: ~33%
Points: ~0.50

COLLAPSE

Details:
All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.

Columns: 2

Sub-Task

100%

Group: Round End
Task #2: Points
Sub Task #1: Show how points are calculated (fastest = higher score)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code show how points are calculated

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works
Response:

1.track correct answers: each player's answer is recorded and checked for correctness in processAnswer(). If correct, the player's ID is saved with a flag indicating a correct answer. 2.get remaining time: when calculating scores, the

Sub-Task

100%

Group: Round End
Task #2: Points
Sub Task #2: Show the code related to syncing points to the Clients

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code related to syncing points to clients

code related to syncing points to the clients part2

end of round ans sync sending the scoreboard to each client

Caption(s) (required) ✓

roundTimer.getRemainingTime() function fetches the remaining seconds in the round. This is key for awarding extra points for faster responses. 3.calculate points: each correct answer is awarded a base score (e.g., 10 points). Additional points are added based on the remaining time (e.g., 1 extra point for each second left). The faster the player answers, the higher their score. 4.award points: for each correct answer, the player's score is incremented by the total points calculated. A log message confirms the awarded points and the player's new total score. 5.end of round: when all players have answered or the timer expires, points are finalized for the round. 6.display scores: the updated scores are broadcasted to all players at the end of the round, showing the leaderboard with points reflecting both correctness and response time.

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1.answer processing:after each player submits an answer, the game checks if it is correct. Each player's answer is recorded in playerAnswers. 2.ending the round:the onRoundEnd() method is triggered when all players answer or the timer runs out.onRoundEnd() calls awardPoints() to handle scoring and syncScoresWithClients() to update all players. 3.point calculation (awardPoints):for each player who answered correctly, awardPoints() adds points (e.g., 10) to their score.this updated score is stored in each player's profile on the server. 4.scoreboard generation (generateScoreboard):generateScoreboard() organizes scores into a formatted list showing each player's name and score.this list is sorted by score in descending order. 5.syncing scores (syncScoresWithClients):syncScoresWithClients() sends the formatted scoreboard to each client.each client receives the updated scoreboard, reflecting their performance. 6.end result:at the end of each round, all players see the updated scoreboard, which shows current scores based on correct answers.

Sub-Task 100%

Group: Round End
Task #2: Points
Sub Task #3: Show the code related to the Client updating their local player map

Sub-Task 100%

Group: Round End
Task #2: Points
Sub Task #4: Show the code related to the in-progress scoreboards

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code related to the client updating their local player map

Task Screenshots

Gallery Style: 2 Columns

4 2 1

method generates the scoreboard, sorting players by score

Syncing Scores with Clients

Caption(s) (required) ✓

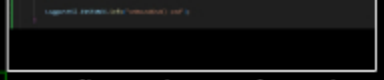
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1.adding players with processClientSync:when a new player joins, processClientSync is triggered.it checks if the player is already in knownClients (the local player map).if not present, it creates a ClientPlayer entry with their clientId and clientName and adds it to knownClients. 2.managing join/elave with processRoomAction:this method handles players joining and leaving the room.if a player joins and isn't in knownClients, it creates and adds a new entry for them.if a player leaves, it removes their entry from knownClients.if the leaving player is the current client, it clears the entire map (knownClients.clear()). 3.overall syncing logic:these methods allow the client to maintain an updated list of players currently in the room.knownClients thus always reflects the current game state by adding new players and removing those who leave.



Awarding Points and Syncing Scores at Round End

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1.generate the scoreboard:at any point, the generateScoreboard method can be called to compile the latest scores.this method sorts players by their scores in descending order, formats each player's name and score, and joins them into a single string. 2.sync scores to clients:the syncScoresWithClients method is used to broadcast the generated scoreboard to all clients.it fetches the latest scoreboard from generateScoreboard and sends it to each client in playersInRoom. 3.end of round actions:when a round ends (onRoundEnd), the game:calls awardPoints to give points to players who answered correctly.calls syncScoresWithClients to send the updated scoreboard to all clients.this ensures that, after every round, clients receive the latest scoreboard showing updated scores and rankings. 4.repeat or end game:after syncing the scores, onRoundEnd checks if the game should continue to a new round or end.if rounds remain, it starts a new round; otherwise, it calls endSession to finish the game.

Sub-Task

100%

Group: Round End

Task #2: Points

Sub Task #5: Show at least 3 examples of terminal output of the scoreboards

Task Screenshots

Gallery Style: 2 Columns

4 2 1

the 3 examples shown between the clients of the scoreboard at the end of the game

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works
Response:

1.start of round:a new question appears on each client's terminal.clients submit answers by typing A, B, C, or D and pressing Enter. 2.answer submission and score updates:as each player submits their answer, the server checks correctness.correct answers are rewarded with points. If your setup involves timing, players who answer faster may earn additional points. 3.score sync on round end:when the round ends (either because time ran out or all players answered), the server compiles the latest scores into a formatted scoreboard, broadcasts the scoreboard to all clients. 4.display on client terminals:each client sees the updated scoreboard, listing each player's name and points. 5.continue or end game:if more rounds remain, a new question will follow in each client's terminal.When all rounds are complete, a "Game Over" message with the final scoreboard appears.

End of Task 2

Task

100%

Group: Round End
Task #3: Next Round or Session End
Weight: ~33%
Points: ~0.50

COLLAPSE

Details:
All code screenshots must have ucid/date visible.

Sub-Task

100%

Group: Round End
Task #3: Next Round or Session End
Sub Task #1: Show the code related to triggering the next round or session end (end game condition)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Code for Triggering Next Round or Session End

endSession() method responsible for finalizing the game and displaying the final scoreboard

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1.end round (onRoundEnd()):award Points: Calls awardPoints() to assign points to players based on their answers. 2.update scores: uses syncScoresWithClients() to send the updated scores to all players. 3.clear round data: resets player answers and increments the currentRound counter, preparing for the next round. 4.check game progress:condition check if currentRound has reached MAX_ROUNDS or no questions are left (questions.isEmpty()), the game session will end. Otherwise, it triggers the next round. 5.end game or start next round:end Game calls endSession() to finalize the game, displaying the final scoreboard and resetting the game state. 6.next round: calls onRoundStart() to start a new round if rounds remain. 7.end session (endSession()):broadcast Final Scores sends a final scoreboard to all players. 8.reset game state: resets player data and scores and changes the game phase to READY, signaling that players can start a new session if desired.

End of Task 3

End of Group: Round End
Task Status: 3/3

Group

100%

Group: Session End
Tasks: 3
Points: 1.25

COLLAPSE

Task

100%

Group: Session End
Task #1: Session End Condition
Weight: ~33%
Points: ~0.42

COLLAPSE

Details:
All code screenshots must have ucid/date visible.

Sub-Task

100%

Group: Session End
Task #1: Session End Condition
Sub Task #1: Show the code related to the session ending when X rounds have passed

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
@Override
protected void onRoundEnd() {
    LoggerUtil.INSTANCE.info("onRoundEnd() start");

    awardPoints(); //vvh - 11/11/24 Awards points to players based on answer correctness
    syncScoresWithClients(); //vvh - 11/11/24 Updates all clients with the latest scoreboard
    playerAnswers.clear();
    currentRound++; //vvh - 11/10/24 Increment round counter

    if (currentRound >= MAX_ROUNDS || questions.isEmpty()) {
        endSession();  // End the session directly
    } else {
        onRoundStart();
    }

    LoggerUtil.INSTANCE.info("onRoundEnd() end");
}
```

code related to ending the session after a set number of rounds

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1.end round processing:the onRoundEnd() method is called at the end of each round.points are awarded to players, scores are synced with clients, and player answers are cleared. 2.round count check:the code checks if currentRound has reached MAX_ROUNDS.if MAX_ROUNDS is reached (or if no more questions are left), it proceeds to end the session by calling endSession(). 3.ending the session:if the session needs to end, endSession() is invoked.otherwise, onRoundStart() is called to begin the next round.

End of Task 1

Task

100%

Group: Session End
Task #2: Scoreboards
Weight: ~33%
Points: ~0.42

COLLAPSE

Details:
All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.

Columns: 2

Sub-Task 100%
Group: Session End
Task #2: Scoreboards
Sub Task #1: Show the code related to the final scoreboard

Sub-Task 100%
Group: Session End
Task #2: Scoreboards
Sub Task #2: Show the final scoreboard from the terminal

Task Screenshots

Gallery Style: 2 Columns

4 2 1

scoreboard is generated and sent to all players when the session ends

generate scoreboard code

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Screenshots

Gallery Style: 2 Columns

4 2 1

finalscoreboards between 3 clients

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1.generate final scoreboard:when the game session ends, the endSession() method is triggered.inside endSession(), the generateScoreboard() method is called to compile the final scores. generateScoreboard() sorts players by their scores in descending order, formats each player's name and score, and joins them into a single scoreboard string. 2.broadcast final scoreboard:after generating the scoreboard, endSession() creates a message prefixed with "Game Over! Final Scoreboard" and includes the sorted player scores.this message is then sent to every player in the game using sendMessage(), so each client receives the final results. 3.prepare for next session:after sending the final scoreboard to all players, endSession() resets game data, clears player scores, and sets the game phase to READY. this reset allows players to start a new session when they're ready, ensuring the game is prepared for a fresh start.

End of Task 2

Task

100%

Group: Session End
Task #3: Cleanup
Weight: ~33%
Points: ~0.42

COLLAPSE

Details:
All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.

Columns: 3

Sub-Task

100%

Group: Session End
Task #3: Cleanup
Sub Task #1: Show the code related to resetting the Player data client-side and server-side (do not

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Server-Side: Resetting Player Data in GameRoom

Client-Side: Resetting Player Data on Client

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1.session end:when the game

Sub-Task

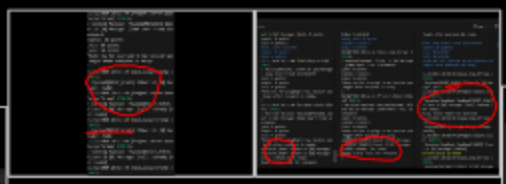
100%

Group: Session End
Task #3: Cleanup
Sub Task #2: Show any terminal evidence

Task Screenshots

Gallery Style: 2 Columns

4 2 1



terminal output from server side

shows how after the game ends, it resets the score board

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Session End
Task #3: Cleanup
Sub Task #3: Show the code related to shifting back to the ready Phase

Task Screenshots

Gallery Style: 2 Columns

4 2 1

GameRoom Code for Shifting Back to Ready Phase

GameRoom Code for Shifting Back to Ready Phase part2

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

1.session end trigger: when a session end condition is met (e.g.,

session ends (after a certain number of rounds or specific conditions), the endSession() method in GameRoom is triggered. 2.server-side reset:the endSession() method calls resetGameData(), which iterates over all players in the game room. 3.resetGameData():calls resetScore() on each player to set their score back to zero.clears any round-specific data, such as recorded answers and the round counter. 4.client notification:the server sends a RESET_READY payload to all connected clients, indicating the session has ended and data has been reset. 5.client-side reset:upon receiving RESET_READY, the client calls processResetReady() to:reset any client-side indicators of readiness.optionally clear UI or local score displays. 6.player connections maintained:players remain connected in the game room, ready for a new session. They don't get moved to a lobby or disconnected; only their game-specific data (scores, answers, readiness) is reset.

max rounds reached), endSession() is called. 2.display final scoreboard:the server compiles and sends a final scoreboard message to all players, showing their scores. 3.reset game data:resetGameData() clears player scores, stored answers, and the round counter to prepare for a fresh start in the next session. 4.reset ready status:resetReadyStatus() sends a reset message to all players, marking them as "not ready" to start the new session. 5.switch to ready phase:changePhase(Phase.READY) updates the game state to "READY" and notifies all players of this phase change. 6.logging:a log entry confirms the transition back to the "READY" phase, indicating readiness for a new session.

End of Task 3

End of Group: Session End
Task Status: 3/3

Group

100%

Group: Misc
Tasks: 3
Points: 1

COLLAPSE

Task

100%

Group: Misc
Task #1: Add the pull request link for the branch
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:
Note: the link should end with /pull/#

Task URLs

URL #1
https://github.com/vvh24/vvh-IT114-003/pull/12

url
https://github.com/vvh24/vvh-IT114-003/pull/12

End of Task 1

Task

100%

Group: Misc
Task #2: Talk about any issues or learnings during this assignment
Weight: ~33%
Points: ~0.33

COLLAPSE

Task Response Prompt

Response:

Building this program was definitely an interesting and sometimes challenging. One of the main issues I encountered was getting the server and client communication to sync seamlessly, especially for real-time events like timing responses and keeping players' scores updated. There were moments when the countdown timer seemed to be inconsistent between the client and server, and it took some deep dives into how callbacks and timers were implemented to understand where the lag was happening. debugging these issues wasn't always straightforward, as each modification required thorough testing to make sure that it didn't break the flow for any connected clients. This part was a bit stressful, as I wanted to ensure that every player received the same updates simultaneously and accurately.Another key learning was how to structure the game's session flow and end conditions, managing how the system reset player data without disconnecting them or sending them back to the lobby.

End of Task 2

Task

100%

Group: Misc
Task #3: WakaTime Screenshot
Weight: ~33%
Points: ~0.33

COLLAPSE

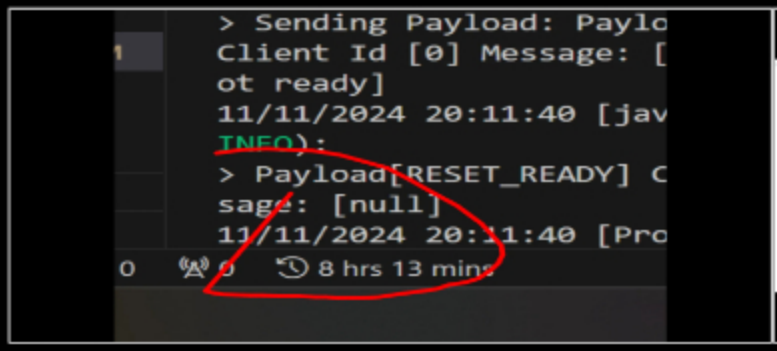
Details:
Grab a snippet showing the approximate time involved that clearly shows your repository. The

Grab a snippet showing the approximate time involved that clearly shows your repository. The duration isn't considered for grading, but there should be some time involved

Task Screenshots

Gallery Style: 2 Columns

4 2 1



wakatime in vscode

overall time in folder

time spent on files

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment